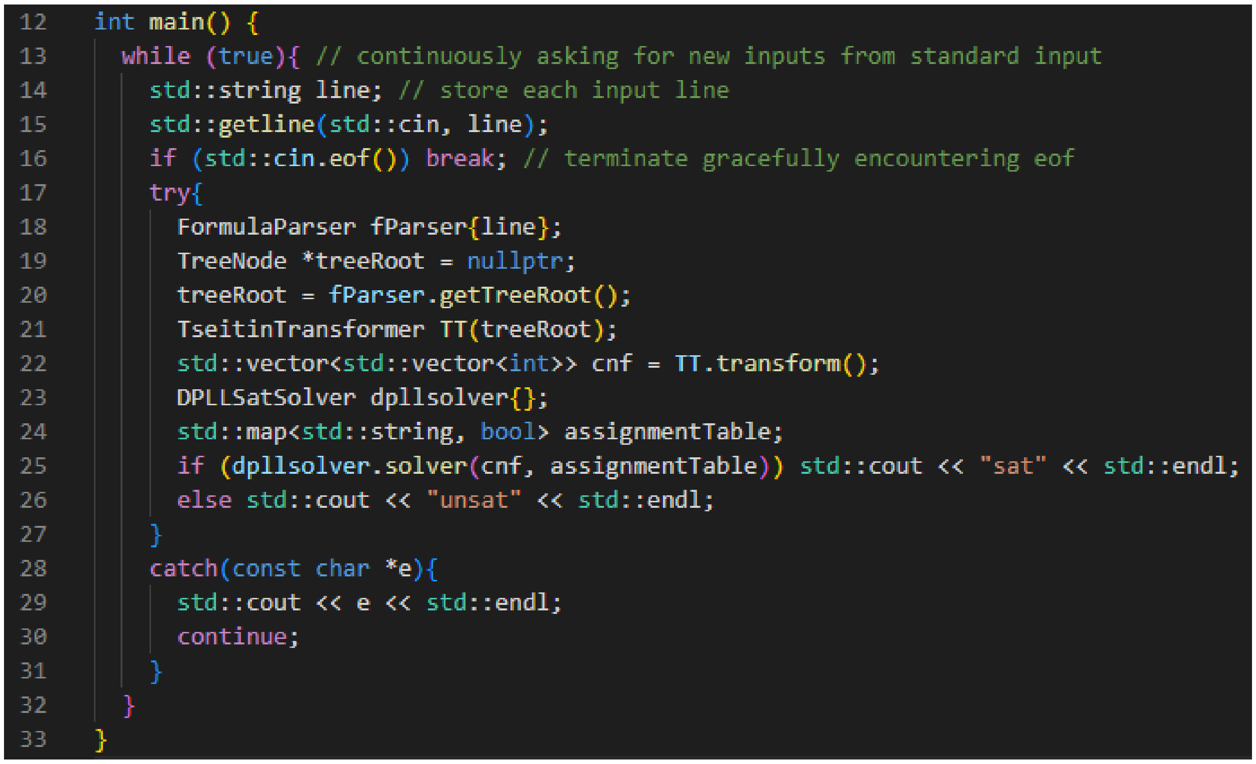
ECE 650 Bonus Programming Assignment Description

Name: Gabriel Chen

WatID: s65chen

ID: 21029699

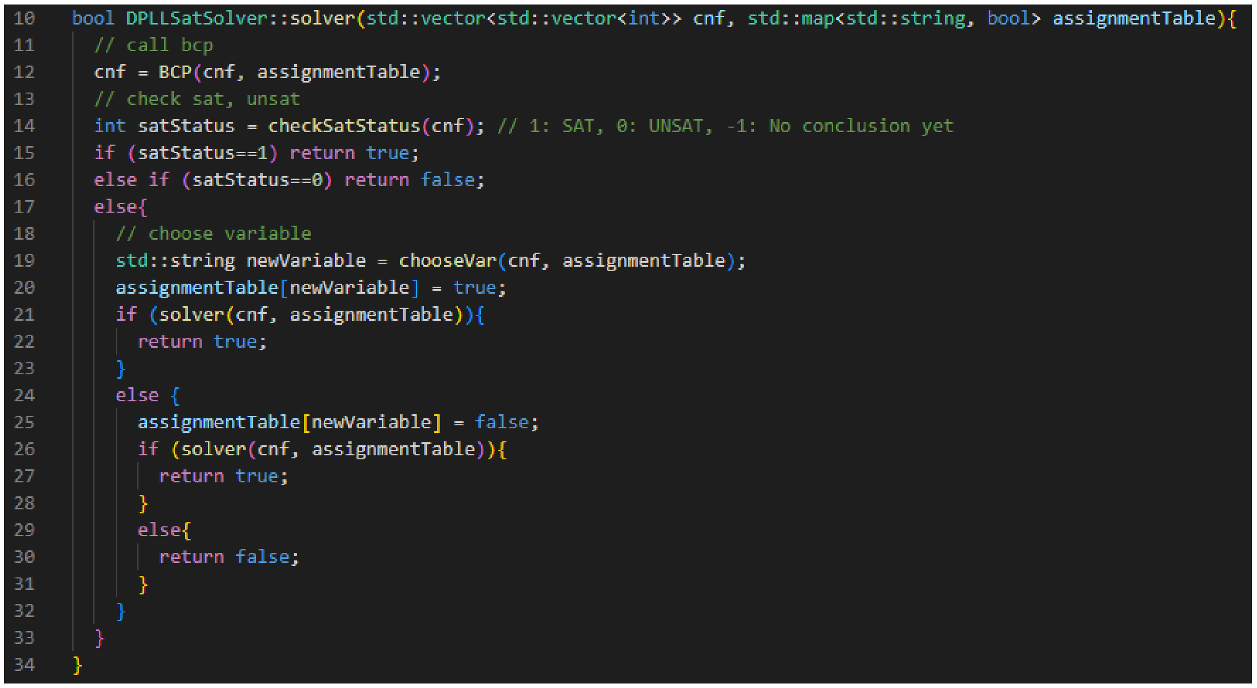
A. In main function:



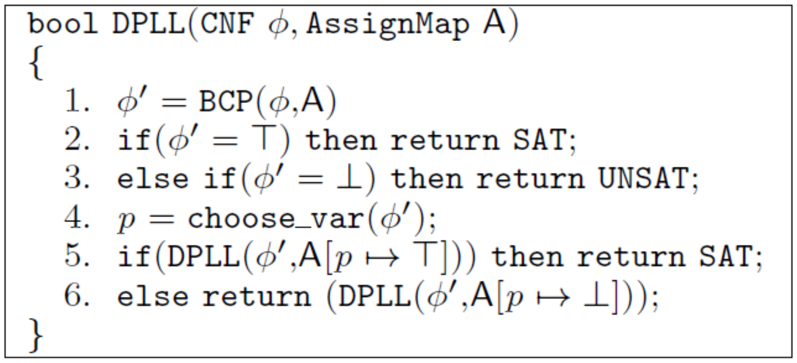
```
12  int main() {
13    while (true){ // continuously asking for new inputs from standard input
14      std::string line; // store each input line
15      std::getline(std::cin, line);
16      if (std::cin.eof()) break; // terminate gracefully encountering eof
17      try{
18        FormulaParser fParser{line};
19        TreeNode *treeRoot = nullptr;
20        treeRoot = fParser.getTreeRoot();
21        TseitinTransformer TT(treeRoot);
22        std::vector<std::vector<int>> cnf = TT.transform();
23        DPLLSatSolver dpllsolver{};
24        std::map<std::string, bool> assignmentTable;
25        if (dpllsolver.solver(cnf, assignmentTable)) std::cout << "sat" << std::endl;
26        else std::cout << "unsat" << std::endl;
27      }
28      catch(const char *e){
29        std::cout << e << std::endl;
30        continue;
31      }
32    }
33  }
```

a. Line 18 - 20: Initialize parser, get the tree node.
b. Line 21 – 23: get the Tseitin transformed CNF
c. Line 24 – 26: call DPLL solver, return “sat” if input formula is satisfiable, else return “unsat”
d. Line 28 – 31: print thrown exception.

B. In solver(), dpllsolver.cc



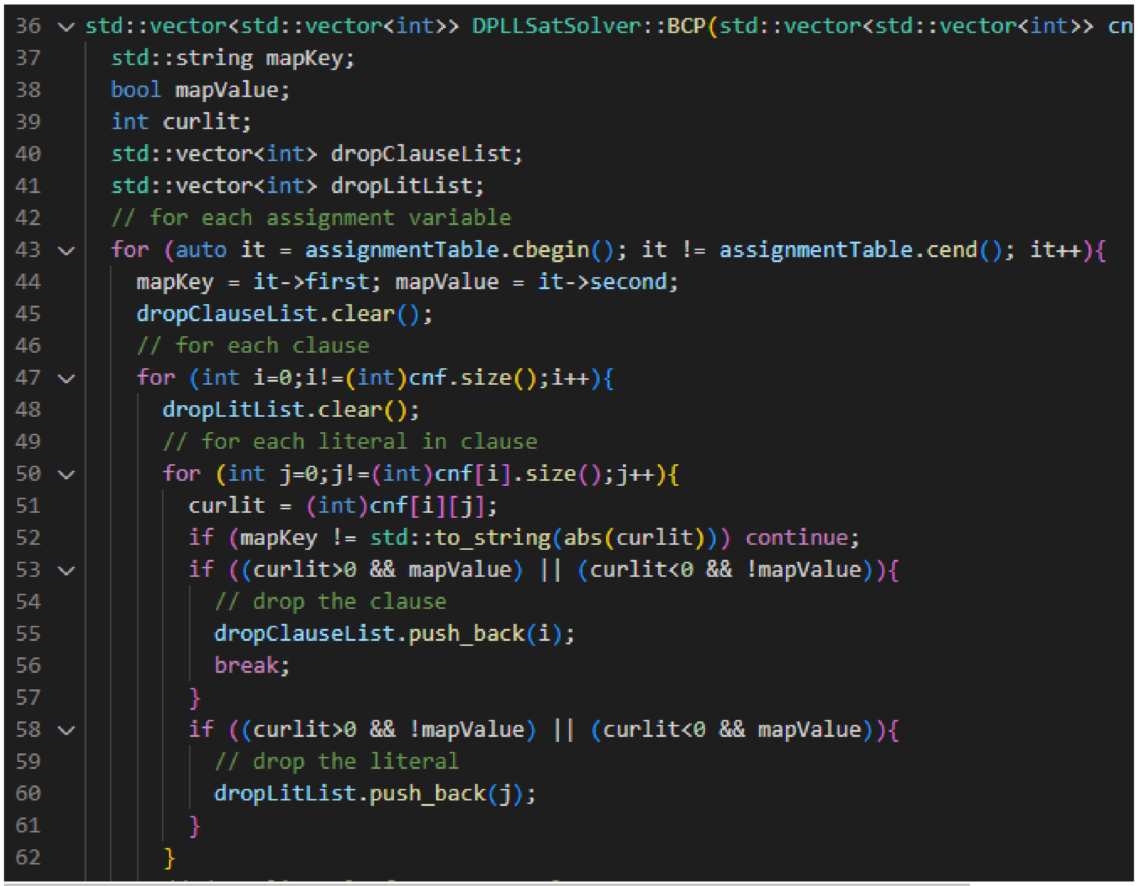
```
10  bool DPLLSatSolver::solver(std::vector<std::vector<int>> cnf, std::map<std::string, bool> assignmentTable){
11    // call bcp
12    cnf = BCP(cnf, assignmentTable);
13    // check sat, unsat
14    int satStatus = checkSatStatus(cnf); // 1: SAT, 0: UNSAT, -1: No conclusion yet
15    if (satStatus==1) return true;
16    else if (satStatus==0) return false;
17    else{
18      // choose variable
19      std::string newVariable = chooseVar(cnf, assignmentTable);
20      assignmentTable[newVariable] = true;
21      if (solver(cnf, assignmentTable)){
22        return true;
23      }
24      else {
25        assignmentTable[newVariable] = false;
26        if (solver(cnf, assignmentTable)){
27          return true;
28        }
29        else{
30          return false;
31        }
32      }
33    }
34  }
```



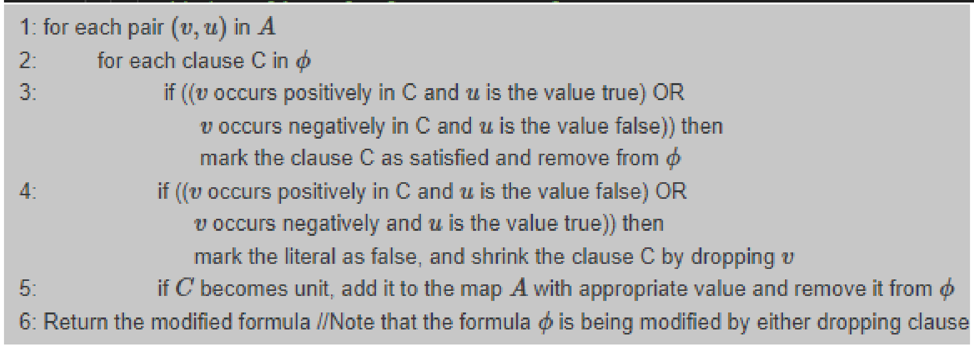
```
bool DPLL(CNF $\phi$, AssignMap A)
{
    1. $\phi' = \text{BCP}(\phi, A)$
    2. if($\phi' = \top$) then return SAT;
    3. else if($\phi' = \bot$) then return UNSAT;
    4. $p$ = choose_var($\phi'$);
    5. if(DPLL($\phi'$,A[$p \mapsto \top$])) then return SAT;
    6. else return (DPLL($\phi'$,A[$p \mapsto \bot$]));
}
```

a. In my code, line 12 is implementing BCP in pseudo code line 1
b. In my code, line 14-16 is implementing pseudo code line 2, 3. The function checkSatStatus() will return “1” if current CNF is satisfiable, return “0” for unsatisfiable, return “-1” if no conclusion yet.
c. In my code, line 19 is implementing pseudo code line 4, choose first variable in CNF which is not in assignment table yet.
d. In my code, line 20 - 23 is implementing pseudo code line 5, recursively run solver() with new assignment as true, return true if it is satisfiable.
e. In my code, line 24 - 31 is implementing pseudo code line 6, if previous assignment failed, run solver again with assignment as false, and return accordingly.

C. In BCP (part 1):



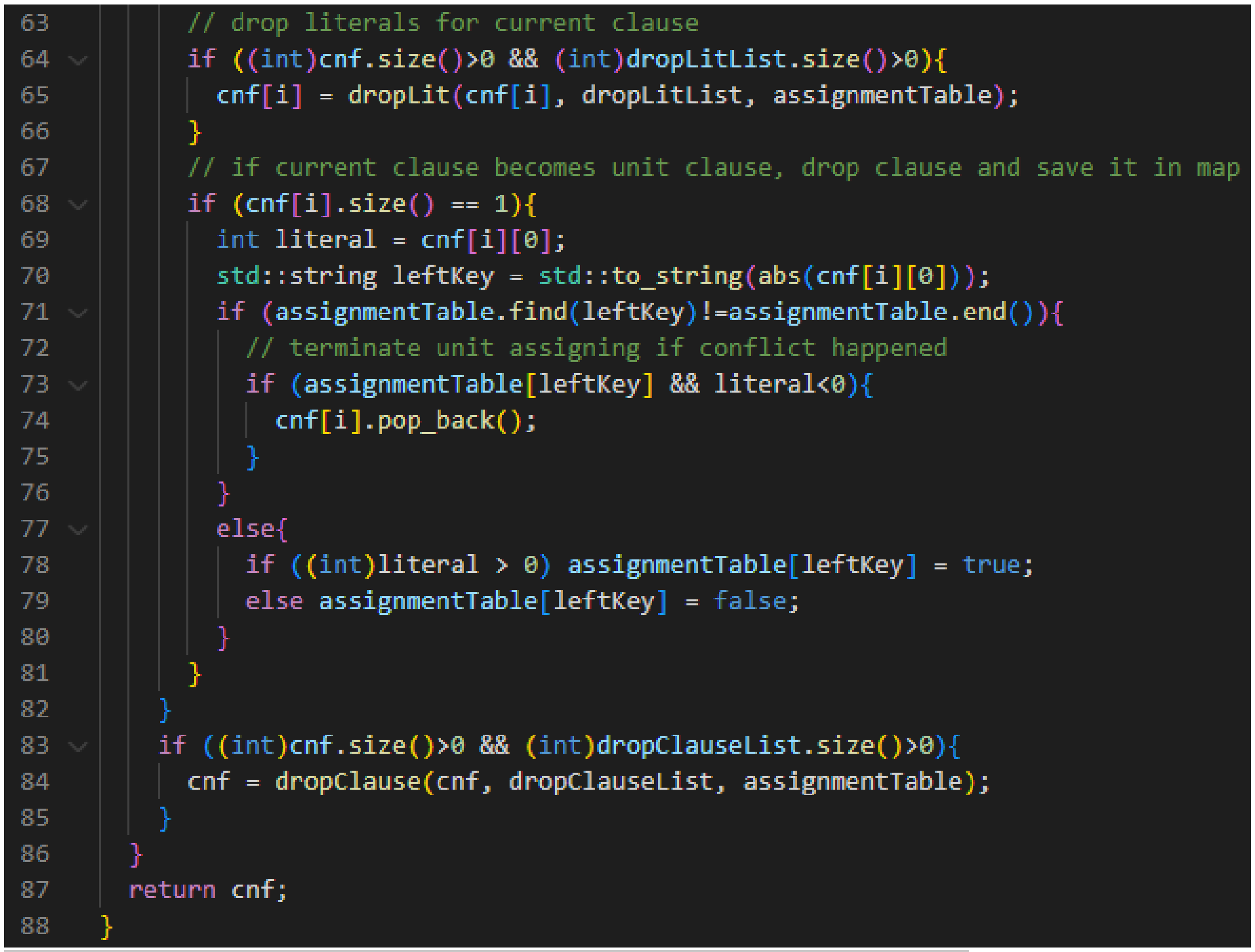
```
36  std::vector<std::vector<int>> DPLLSatSolver::BCP(std::vector<std::vector<int>> cn
37      std::string mapKey;
38      bool mapValue;
39      int curlit;
40      std::vector<int> dropClauseList;
41      std::vector<int> dropLitList;
42      // for each assignment variable
43      for (auto it = assignmentTable.cbegin(); it != assignmentTable.cend(); it++){
44          mapKey = it->first; mapValue = it->second;
45          dropClauseList.clear();
46          // for each clause
47          for (int i=0;i!=(int)cnf.size();i++){
48              dropLitList.clear();
49              // for each literal in clause
50              for (int j=0;j!=(int)cnf[i].size();j++){
51                  curlit = (int)cnf[i][j];
52                  if (mapKey != std::to_string(abs(curlit))) continue;
53                  if ((curlit>0 && mapValue) || (curlit<0 && !mapValue)){
54                      // drop the clause
55                      dropClauseList.push_back(i);
56                      break;
57                  }
58                  if ((curlit>0 && !mapValue) || (curlit<0 && mapValue)){
59                      // drop the literal
60                      dropLitList.push_back(j);
61                  }
62              }
```



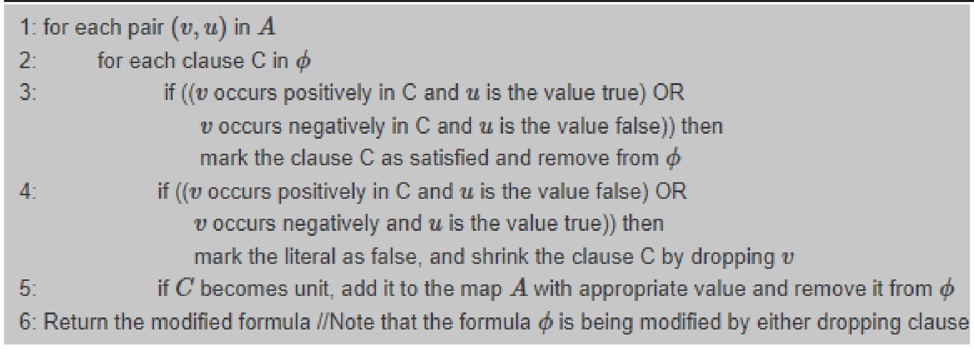
```
1: for each pair $(v, u)$ in $A$
2:     for each clause C in $\phi$
3:         if (($v$ occurs positively in C and $u$ is the value true) OR
               $v$ occurs negatively in C and $u$ is the value false)) then
               mark the clause C as satisfied and remove from $\phi$
4:         if (($v$ occurs positively in C and $u$ is the value false) OR
               $v$ occurs negatively and $u$ is the value true)) then
               mark the literal as false, and shrink the clause C by dropping $v$
5:         if $C$ becomes unit, add it to the map $A$ with appropriate value and remove it from $\phi$
6: Return the modified formula //Note that the formula $\phi$ is being modified by either dropping clause
```

a. In my code line 43 is implementing pseudo code line 1, going through every assignment in the table.
b. In my code line 47 is implementing pseudo code line 2, going through every clause.
c. In my code line 50 – 57 is implementing pseudo code line 3, finding SAT clause and put current clause into the drop clause list. (we will drop them in the following BCP part 2)
d. In my code line 58 – 60 is implementing pseudo code line 4, finding UNSAT literal and put current literal into the drop literal list. (we will drop them in the following BCP part 2)

D. In BCP (part 2):



```
63        // drop literals for current clause
64        if ((int)cnf.size()>0 && (int)dropLitList.size()>0){
65          cnf[i] = dropLit(cnf[i], dropLitList, assignmentTable);
66        }
67        // if current clause becomes unit clause, drop clause and save it in map
68        if (cnf[i].size() == 1){
69          int literal = cnf[i][0];
70          std::string leftKey = std::to_string(abs(cnf[i][0]));
71          if (assignmentTable.find(leftKey)!=assignmentTable.end()){
72            // terminate unit assigning if conflict happened
73            if (assignmentTable[leftKey] && literal<0){
74              cnf[i].pop_back();
75            }
76          }
77          else{
78            if ((int)literal > 0) assignmentTable[leftKey] = true;
79            else assignmentTable[leftKey] = false;
80          }
81        }
82      }
83      if ((int)cnf.size()>0 && (int)dropClauseList.size()>0){
84        cnf = dropClause(cnf, dropClauseList, assignmentTable);
85      }
86    }
87    return cnf;
88  }
```



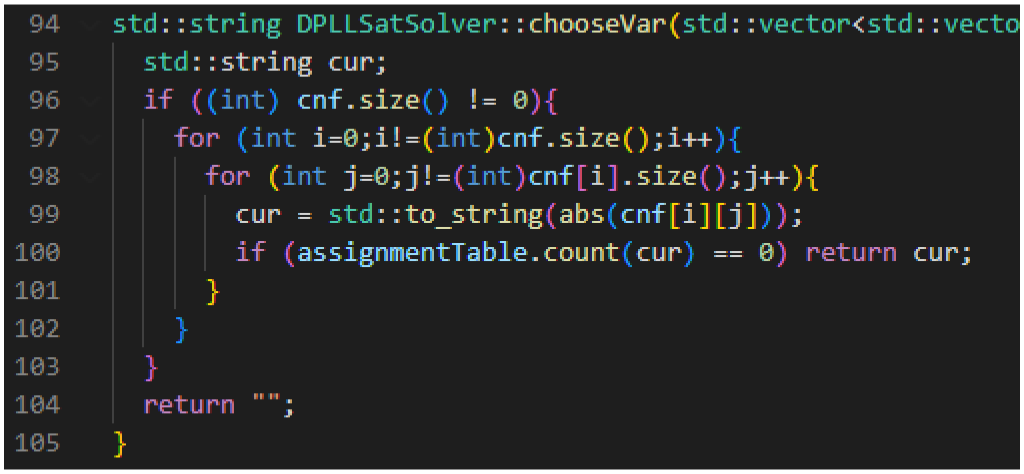
```
1: for each pair $(v, u)$ in $A$
2:     for each clause C in $\phi$
3:         if (($v$ occurs positively in C and $u$ is the value true) OR
               $v$ occurs negatively in C and $u$ is the value false)) then
               mark the clause C as satisfied and remove from $\phi$
4:         if (($v$ occurs positively in C and $u$ is the value false) OR
               $v$ occurs negatively and $u$ is the value true)) then
               mark the literal as false, and shrink the clause C by dropping $v$
5:         if $C$ becomes unit, add it to the map $A$ with appropriate value and remove it from $\phi$
6: Return the modified formula //Note that the formula $\phi$ is being modified by either dropping clause
```

a. In my code line 64 – 66 is implementing pseudo code line 4, when the loop for current clause is done, we start to drop literals here.

b. In my code line 68 – 81 is implementing pseudo code line 5, if the current clause becomes unit clause, (line 78 - 79) we put the only literal in the unit clause into the assignment table accordingly (so the clause will be drop later). (line 71 - 76) If there is a conflict with the existing table, drop the literal in the clause (so it will be identified as UNSAT later)

   c. In my code line 83 – 85 is implementing pseudo code line 3, when all of the clauses are done for the current variable, start dropping clauses.
   d. In my code line 87 is implementing pseudo code line 6, returning modified CNF.

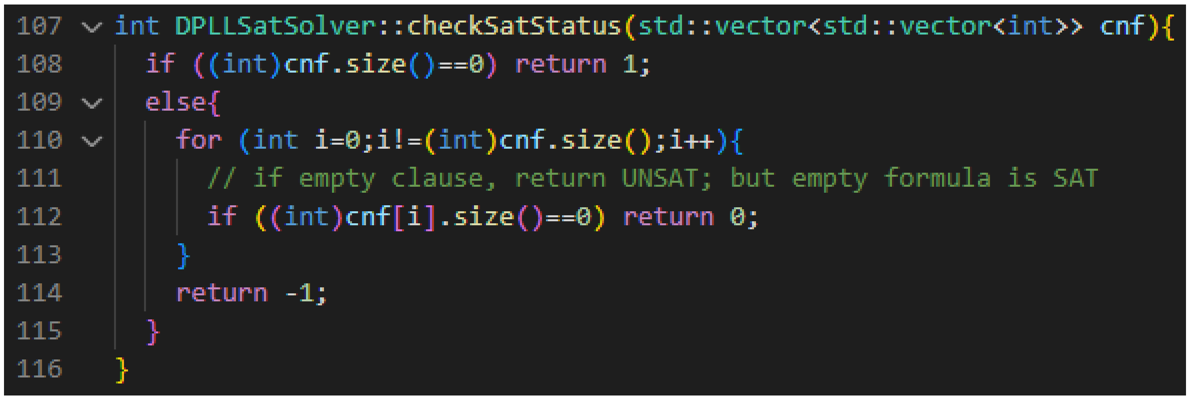
E. In function chooseVar():



```cpp
std::string DPLLSatSolver::chooseVar(std::vector<std::vecto
    std::string cur;
    if ((int) cnf.size() != 0){
        for (int i=0;i!=(int)cnf.size();i++){
            for (int j=0;j!=(int)cnf[i].size();j++){
                cur = std::to_string(abs(cnf[i][j]));
                if (assignmentTable.count(cur) == 0) return cur;
            }
        }
    }
    return "";
}
```

   a. Get the first variable from CNF which is not in the assignment table and return it.

F. In function checkSatStatus():



```cpp
int DPLLSatSolver::checkSatStatus(std::vector<std::vector<int>> cnf){
    if ((int)cnf.size()==0) return 1;
    else{
        for (int i=0;i!=(int)cnf.size();i++){
            // if empty clause, return UNSAT; but empty formula is SAT
            if ((int)cnf[i].size()==0) return 0;
        }
        return -1;
    }
}
```

   a. If CNF is empty, return 1 (SAT); If there is any clause in CNF that is empty, return 0 (UNSAT); The rest of the case return -1 (no conclusion yet)